At The Beginning…

Heaven is compiled line by line. There is no need for any semicolons, which is why they have been omitted. To convert heaven code to java code and execute it, use the **initial** method from the **Initializing** class.

Datatypes

Type	Refers To
text	String
num	Integer
lnum	Long
fnum	Float
truth	Boolean
listastext	ArrayList<String>
listasnum	ArrayList<Integer>
listaslnum	ArrayList<Long>
listasfnum	ArrayList<Float>
listastruth	ArrayList<Boolean>

null and **void** keywords are represented with **empty**.

Variable Assigning

Allowed patterns:

@type var = empty
@type var = val
@type var1, var2, var3 = val1, val2, val3

Multiple assigning at once is allowed. Also, global variables are assigned with the same syntax.

Functions

Main Function

$main
//code
$done

Void Function

$empty name @para
//code
$done

Other Functions

$type name @para
//code
$return val

If the value is text and represented with a variable name, the underscore mark (_) must be put before it. Except, there is no need to use double quotes for texts.

Loop Statements

Instead of **for** and **while** keywords, only **loop** is used in Heaven.

For Loop

loop type var = val; condition; iteration
//code
end

While Loop

loop condition
//code
end

For-Each Loop

loop type var in array
//code
end

Conditional Statements

If

if condition
//code
end

Else If

elif condition
//code
end

Else

else condition
//code
end

Switch - Case

acc means **according to**.

acc var
let val1
//code
block
let val2
//code
block
let val3
//code
block
finish

Lists

Allowed patterns:

@type list1 = {item1, item2, item3}
@type list1, list2, list3 = {item1, item2, item3} {item4, item5, item6} {item7, item8, item9}

If the type is **text,** there are some extra rules:

- Text values are written without quote marks.
- To put text values, if they include, comma (,) and closing curly bracket (}) must be written with ‘ \ ’ symbol.
- If a variable will be added instead of text value, the underscore mark (_) is put on its name.

IO Library

IO library is used for basic input - output functionality.

Output

io >> out “val”
io >> out var

Input

io >> in = @type var
io >> in = var

Lists Library

Lists library provides basic functions for lists.

Set Item

lists >> set at index in list _var (for any type)
lists >> set at index in list val (no need for double quote marks)

Get Item

lists >> get at index in list = @type var
lists >> get at index in list = var

Add Item

lists >> add in list _var (for any type)
lists >> add in list val (no need for double quote marks)

Remove Item

lists >> remove at index in list

Importing

Libraries can be imported by using this pattern:

/call
libraryName1
libraryName2
libraryName3
/them

Files Library

Writing

files >> write path var

path can be either **val** or **_var**. If path is val then it should not contain these characters:

- white space
- @
- “
- ,
- {
- }
- \
- $
- >
- <
- ;

var type must be **listastext**.

Reading

These patterns are allowed:

files >> read path

path must be **text val** or **_var**. This method prints out the console.

files >> read path = var
files >> read path = @type var

The type of variable must be **text**.

Automated Variables

These variables are created automatically during translation from heaven code to java code. They have **var + 8 digits** name pattern. Their sequence starts with **var00000000**, ends with **var99999999**. Which is why naming in this way is not suggested.

Math Library

Random

math >> random from val to val = var
math >> random from val to var = var
math >> random from var to val = var
math >> random from var to var = var

math >> random from val to val = @type var
math >> random from val to var = @type var
math >> random from var to val = @type var
math >> random from var to var = @type var

Type must be **num**. **from** value is inclusive, **to** value is exclusive.

Exponent

math >> pow val val = var
math >> pow val var = var
math >> pow var val = var
math >> pow var var = var

math >> pow val val = @type var
math >> pow val var = @type var

math >> pow var val = @type var
math >> pow var var = @type var

First parameter is **base**, the second one is **exponent**. The method returns **num** and takes **num** parameters.

Root

math >> root val val = var
math >> root val var = var
math >> root var val = var
math >> root var var = var

math >> root val val = @type var
math >> root val var = @type var
math >> root var val = @type var
math >> root var var = @type var

First parameter is **base**, the second one is **root**. The method returns **fnum** and takes **fnum** parameters.

Absolute

math >> abs var = var
math >> abs val = var
math >> abs var = @type var
math >> abs val = @type var

The method returns **fnum**. Also, takes **fnum** as parameter.

Sorting

math >> order var

It takes **listasnum**, **listasfnum** and **listaslnum** as parameter. It sorts items from the least to the greatest.

math >> revOrder var

Unlike **order**, **revOrder** sorts them from the greatest to the least.

Type Casting

That process is made by using **/type** command. These patterns are allowed in type casting:

/type var1 as type = var2
/type var1 as type = @new var2

GUI Library

UI Creating

This pattern is allowed to create a frame:

gui >> launch varTypes varNames varFrames frameNames

varTypes, **varNames** and **frameNames** are **listastext** and **varFrames** is **listasnum**. **varTypes** must include Swing component names. It is not recommended to use *frame*, *color* and *isVisible* names inside **varNames** and **frameNames**. **varFrames** must contain indexes of frame objects inside **frameNames**.